Supplementary Material
Functionality understanding and segmentation in 3D scenes

Jaime Corsetti[1,2] Francesco Giuliari[1] Alice Fasoli[1] Davide Boscaini[1] Fabio Poiesi[1]

[1]Fondazione Bruno Kessler [2]University of Trento

{jcorsetti,fgiuliari,alfasoli,dboscaini,poiesi}@fbk.eu

In this supplementary material, we provide an analysis of the methods used for functionality segmentation (Sec. 1), delve deeper into the ablation studies highlighting the capabilities of our proposed approach (Sec. 2), and present additional qualitative results (Sec. 3). We finally report the hardware setup and the implementation details in Sec. 4.

1. Baselines

In the following section, we report the architecture of the baseline methods, as well as their implementation details (Sec. 1.1). We then show the results obtained with variations in the prompt and architecture of the baselines (Sec. 1.2).

1.1. Methods details

As SceneFun3D did not release the code used for OpenMask3D and LERF, we use their original code and follow SceneFun3D's instructions to reproduce the results.

OpenMask3D [11] is an open-vocabulary 3D instance segmentation method that operates jointly on 3D and 2D data. As first step, Mask3D [10] is used on the scene to obtain a set of 3D class-agnostic masks. The 3D masks are projected on the scene views, and the top K views are selected according to the visibility of the projected mask on the frames. The selected views are segmented via SAM [7] and CLIP [9] is used to extract multi-scale embedding from each SAM mask. By leveraging 2D-3D correspondences, the features for each mask are aggregated in 3D, so that a CLIP feature vector characterizes each original 3D mask. Open-vocabulary segmentation is obtained by embedding the description `D` with CLIP and retrieving the most similar masks, selecting the ones with a similarity higher than a threshold τ.

Following SceneFun3D [4], we test OpenMask3D without retraining, with the original implementation[1] and checkpoint trained on ScanNet200 [2]. To account for the high dimension of SceneFun3D point clouds with respect to the ones used for training, we subsample each point cloud to a maximum of 2 million points and retain the top 200 masks for the 2D segmentation procedure.

OpenIns3D [5] is a 3D-only framework for 3D open-vocabulary instance segmentation. It features three modules, named 'Mask', 'Snap', and 'Lookup'. The 'Mask' module generates class-agnostic mask proposals in 3D point clouds, while the 'Snap' module generates synthetic images of the scene at multiple scales, in order to cover the locations of the previously extracted 3D masks. The generated views are processed by an open-vocabulary 2D localization module [12] to extract the objects from the textual description `D`. Finally, the 'Lookup' module searches through the outcomes of each extracted 2D mask, and by using Mask2Pixels correspondences transfers the semantic labels from 2D to 3D. We test OpenIns3D on SceneFun3D without retraining, using the official implementation[2] and the checkpoint trained on ScanNet200 [2]. However, we found that using the 'Snap' module is not beneficial in our case (see Tab. 2), and instead we use the 2D images provided by SceneFun3D. Specifically,

Table 1. Results obtained by OpenMask3D, LERF, and OpenIns3D with different types of prompts, on split0 of SceneFun3D.

	Method	Prompt	mAP	AP_{50}	AP_{25}	mAR	AR_{50}	AR_{25}	mIoU
1	OpenMask3D [11]	original	0.2	0.2	0.4	20.3	24.5	27.0	0.2
2		parsed	0.2	0.2	0.2	13.9	17.1	19.8	0.2
3	OpenIns3D [5]	original	0.0	0.0	0.0	40.5	46.7	51.5	0.1
4		parsed	0.0	0.0	0.0	30.9	35.5	38.7	0.1
5	LERF [6]	original	0.0	0.0	0.0	34.2	35.1	36.0	0.0
6		parsed	0.0	0.0	0.0	34.2	35.1	35.9	0.0
7	Fun3DU (ours)	original	7.6	16.9	33.3	27.4	38.2	46.7	15.2

Table 2. Results obtained by OpenIns3D [5], with and without the original 'Snap' rendering module used for 2D segmentation, and with different segmentors, on split0 of the SceneFun3D dataset.

	Snap	Segmentor	mAP	AP_{50}	AP_{25}	mAR	AR_{50}	AR_{25}	mIoU
1		Yolo-world [1]	0.0	0.0	0.0	40.5	46.7	51.5	0.1
2	✓	Yolo-world [1]	0.0	0.0	0.0	4.1	4.3	4.9	0.0
3		ODISE [12]	0.0	0.0	0.0	24.8	29.8	31.9	0.0
4	✓	ODISE [12]	0.0	0.0	0.0	4.1	5.6	6.7	0.0

[1] https://github.com/OpenMask3D/openmask3d

[2] https://github.com/Pointcept/OpenIns3D

we sample 120 views per scene at uniform time intervals, and render 24 views when using the 'Snap' module, following the OpenIns3D's standard setting. We found that rendering more than 24 views did not lead to better performance.

OpenIns3D reported results with both Yolo-world [1] and ODISE [12] as 2D detector. We use Yolo-world in our standard setting for better performance. The open-vocabulary 2D detector is used together with CLIP for ranking and filtering the masks, in order to reduce the false positives.

LERF [6] is a method based on neural radiance fields [8] that produces pixel-level feature maps from arbitrary viewpoints within a scene. The features are trained to align with CLIP [9], so that a textual description can be used to obtain the most relevant pixels. By lifting and aggregating the pixels on the 3D point cloud, open-vocabulary 3D semantic segmentation can be performed. For each high-resolution image sequence in SceneFun3D, we train a LERF model using the official implementation[3] integrated with Nerfstudio[4]. We first sample each high-resolution video sequence at 2Hz, and run Colmap to estimate and refine the camera poses. Next, we train LERF models on sequences in which Colmap matches at least 10% of the views and identifies at least 10 valid views. Finally, we use the task description as the prompt to extract relevancy maps for each view. The final 3D mask is obtained by lifting and accumulating the score of each pixel on each point, and finally applying a threshold on points with a positive score.

1.2. Additional results

In Tab. 1 we report the extended results on the baselines, obtained with the prompt composed by concatenating the contextual and functional object names (e.g., '`cabinet handle`'). In principle, using a shorter description is beneficial as all the baselines have been tested with relatively short prompts, i.e., describing single objects. Instead, compared to using the standard prompt provided by SceneFun3D, this leads to a worse performance in all cases, with the exception of LERF which remains on par (rows 5 vs 6). In particular, OpenMask3D loses 0.2 AP_{25} and 7.2 AR_{25} (rows 1 vs 2), while OpenIns3D loses 12.8 AR_{25} (rows 3 vs 4). This suggests that the text encoder used by the baselines [9] is to some extent capable of processing more advanced prompts, and therefore removing the context given from the original prompt hinders the performance.

In Tab. 2 we report additional results on OpenIns3D [5], showing how our modifications influenced its performance on SceneFun3D. Row 1 is the version we use as baseline, in which the RGB frames are used in place of the 'Snap' module and Yolo-world [1] replaces ODISE [12] to perform 2D localization. In row 2, we use the original 'Snap' module, which renders a set of views of the scene according to the 3D mask proposals. Using this module results in a large drop in performance, as the AR_{25} loses 46.6 points. We observe that scene renders are designed to focus on the common furniture objects retrieved from the mask proposal module, often from a top-down perspective. This is suboptimal for finding the small functional objects, and therefore OpenIns3D benefits from using the original frames of SceneFun3D, which often show close-up views of the objects. In rows 3 and 4 we replace Yolo-world with ODISE [12], respectively without and with the 'Snap' module. Both configurations fall short of the baseline at row 1. As the authors of OpenIns3D observed, Yolo-world performs better on real images (i.e., without the 'Snap' module), while ODISE performs better on rendered images (i.e., with the 'Snap' module).

[3] https://github.com/kerrj/lerf

[4] https://docs.nerf.studio/

2. Extended ablation studies

In Sec. 2.1 we extend the ablation study on the number of views $\hat{V}$, by showing additional results and reporting how this hyperparameter influences the processing time.

2.1. Hyperparameter sensitivity analysis

We report in Tab. 3 the mIoU values obtained by varying the threshold τ and the number of sampled views $\hat{V}$. Additionally to the values reported in the main paper, we show the change in performance with $\hat{V} = 100$ and $\hat{V} = 200$. We observe that, compared to our standard setting (row 4, $\hat{V} = 50$), sampling $\hat{V} = 100$ views causes a small increment of 0.6 points of mIoU, but almost doubles the processing time per task description (118.4 vs 204.6 seconds). Instead, setting $\hat{V} = 200$ is not beneficial, as it lowers the mIoU of 1.3 points with respect to our baseline with the default $\tau = 0.7$. When so many views are sampled, spurious masks are more easily retrieved, thereby lowering the final performance. As a trade-off between performance and execution time, we set $\hat{V} = 50$ in our standard setting.

Table 3. Results in mIoU on split0 of SceneFun3D, obtained by sampling different numbers of views $\hat{V}$ and using different thresholds τ. The last row reports the average time per task description `D` for retrieving the top views, performing functional object segmentation, and multi-view agreement to obtain the final point clouds. The timings for task description understanding and contextual object segmentation are excluded, as they are constant.

	τ	Number of views $\hat{V}$ used for View Selection							
		2	4	10	20	30	50	100	200
1	0.1	7.1	6.7	6.4	6.2	5.8	5.5	4.8	3.4
2	0.3	7.1	7.6	8.7	9.8	9.8	10.1	10.1	7.6
3	0.5	8.1	9.3	10.8	12.1	12.6	13.1	14.0	11.4
4	0.7	8.1	9.4	10.7	12.9	13.9	15.3	15.9	14.0
5	0.9	8.1	9.6	10.0	10.9	12.6	13.2	13.3	11.1
	Time (s)	28.4	34.7	42.3	60.7	81.9	118.4	204.6	237.6

3. Additional qualitative results

In Sec. 3.1 we report examples of outputs of the task description understanding module, showing how the contextual and functional object names are retrieved. Sec. 3.2 reports examples of view selection, by showing the retrieved contextual masks along with their scores. In Sec. 3.3, we show examples of points extracted with Molmo [3], along with the final segmentation mask provided by SAM [7]. Finally, in Sec. 3.4 we extend the qualitative results on functionality segmentation provided in the main paper.

3.1. Task description understanding

We report in Fig. 1, some examples of task description understanding, where we use the LLM to extract the functional object `F` and contextual object `O` from a given text description `D`. The description is shown in the blue box, and the LLM response is shown in the red box. For the sake of brevity, in the figure we omit the system message "*You are an AI System that has to provide JSON files to a robotic system so that it can interact with our physical world, based on a natural language prompt. In particular, you have to help the robot in identify which object parts it has to interact with to solve particular tasks. Its set of possible actions are [rotate, key_press, tip_push, hook_pull, pinch_pull, hook_turn, foot_push, plug_in, unplug]*", as it is the same for all conversations. For the same reason, we omit also the part of the user message where we ask the LLM to respond with a structured json: "*How do I {`Task description D`}? Respond directly with only the json with the following format. { "task_solving_sequence": a list of strings with the description of what I have to do to accomplish the task described by the prompt, subdivided in subtasks., "acted_on_object": a string with the name of the object part on which I have to act on., "acted_on_object_hierarchy": a list of object parts from the top level object to the object part.* }".

We observe that in most cases, the LLM successfully provides an object hierarchy that is useful for the view selection process. Typically, the first object in the hierarchy, which we use as "contextual object" (`O`), serves this purpose effectively. However, there are some failure cases, specifically (c) and (d). In case (c), the object hierarchy is reversed; an uncommon but occasionally observed behavior. Despite this reversal, the 'telephone' is still correctly identified as the contextual object. In case (d), however, the contextual object is identified as 'ceiling light', which is unsuitable for view selection. This happens because the functional object 'light switch' would more likely be mounted on the wall.

An interesting observation arises in cases (e) and (f), where a small change in the description, from "left" to "right", leads to significantly different responses. These examples highlight how the LLM reasons at different levels of abstraction. In case (e), it assumes that the "left tap" can be acted upon, identifying it as the functional object. Instead, in case (f), it interprets the tap as part of the object hierarchy, identifying the functional object as the "switch" on the tap.

These examples underline the LLM's overall effectiveness in interpreting task descriptions and generating useful object hierarchies, while also highlighting certain limitations in handling ambiguous or nuanced descriptions. Understanding these strengths and weaknesses is crucial for refining its reasoning capabilities and ensuring more consistent performance in diverse scenarios.

3.2. Score-based view selection

We report in Fig. 2 some examples of contextual object masks retrieved by our view selection mechanism, from the highest score (leftmost) to the lowest score (rightmost). For each mask, we report the total score used for ranking S, the detection confidence score S_m, the distance distribution score S_d, and the angle distribution score S_α. In the first row, we can observe an ideal case, in which the first three views show the contextual object (the kitchen range hood) well visible in the images, while the fourth and fifth images only show a portion of the object. Similarly, in the second row the first two images show a close-up view of the nightstand, while the remaining images show a progressively far and occluded mask. Note that the last two images are the ones with the highest detection confidence S_m among the five samples. This shows that only considering the detection confidence could lead to selecting views that are not optimal for functionality segmentation.

In the third row, the TV stand is well-visible and centered in the first three views. In the last two views instead, partial occlusions lead to a lower score, mostly due to the low angle scores S_α. Finally, the last row shows a more complex case, in which multiple instances of the contextual object (window) are present. In this case, the correct masks (the third and fourth) have high S_d and S_α scores, but due to a lower S_m score, they are not ranked in the first positions. This shows that weighting all the scores can provide the correct masks that would be discarded when only the confidence score S_m is considered.

3.3. Functional object segmentation

Fig. 3 shows some examples of functional object segmentation on the scene views. For each column, we report on the left the point(s) provided by Molmo [3], and on the right the mask(s) obtained by SAM [7], using the point(s) as input.

In the first row, the VLM can correctly point to the functional objects (the bottom drawer handles), but due to the limited image resolution, the whole drawer is segmented in the first case. Due to our view selection strategy, in most cases, the contextual object (the nightstand) is clearly visible in the view, as in the second column, and therefore the final 3D mask accurately segments the functional object.

The second row shows a very ambiguous case, in which

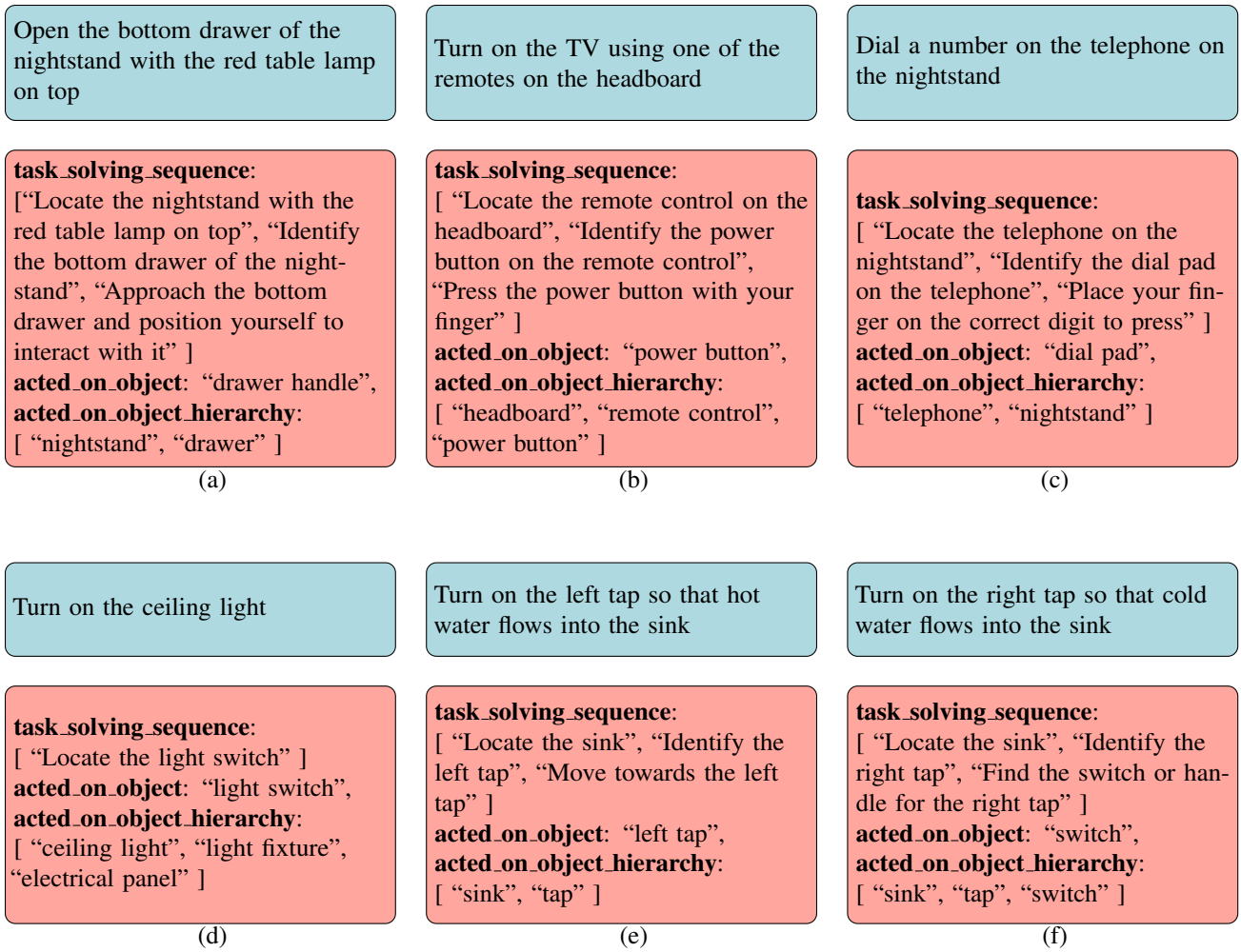


Figure 1. Examples of LLM conversations. The blue box shows the task description D, and the red box is the LLM response. We omit the system message and the JSON structure in the user prompt, which can be observed in Fig. 3 of the main paper.

three radiator parts look similar to the functional object (the dial). In the first case, all objects are pointed and segmented, while in the second case, only the correct one is segmented. These spurious errors are effectively removed by our multi-view agreement strategy, by accumulating the 2D masks and applying a threshold.

In the third row, we can observe a particularly difficult case, in which the functional object (the remote) is very occluded in the first view. Nonetheless, the VLM is capable of pointing to it, resulting in a correct 2D mask. This shows that the VLM is effective not only in finding small objects such as knobs and buttons, but also in recognizing heavily occluded objects, such as the remote in the second case.

In the final row, we observe how the VLM is capable of recognizing the humidifier, which is an unusual object that can have very different appearances based on its design. In both views, the VLM can point to the central button of the humidifier and SAM provides an accurate mask.

3.4. Point cloud segmentation

We report additional qualitative results on split0 of SceneFun3D in Fig. 4. In the first column, Fun3DU is the only method to obtain an accurate segmentation mask of the light switch. Note that the precision in this case is relatively low, as the ground-truth mask only considers the light switch itself, while our method also segments the panel around it. In all the other cases, Fun3DU obtains precision higher than 70 and recall higher than 65, while the baselines either segment the whole contextual object (e.g., OpenIns3D in all cases), or fail in finding relevant points (e.g., OpenMask3D in the third and fourth column, and LERF in the second and fifth column). These examples show particularly small functional objects, which are extremely difficult to handle without ad-hoc techniques as in Fun3DU.

We report qualitative results on split1 of SceneFun3D in Fig. 5. This partition is more difficult than split0, as it fea-

tures more complex scenes with more points, on average. In the first column, Fun3DU can accurately segment the handle on the oven, while other methods fail at finding a relevant mask (OpenMask3D, LERF) or segment a large portion of the kitchen (OpenIns3D). Similarly, in the second column, our method is the only one to locate the small valve at the bottom of the radiator, while OpenIns3D is only capable of segmenting the whole object. The third column shows a case in which Fun3DU fails: instead of segmenting the lock used to open the blue case, it segments the handle just underneath it. This could be due to an error in the task description understanding, in which '`handle`' has been provided by the LLM instead of '`lock`'. Instead, the other methods either segment only the audio system (OpenIns3D) or cannot find any relevant mask. In the fourth column our method correctly identifies the telephone on the TV, but segments the whole phone instead of the dial pad, resulting in high recall and relatively low precision. Finally, the fifth column is a particularly difficult case, in which the task description requires adjusting the seat height of the exercise bike, on which two knobs are located. Fun3DU segments both knobs, still resulting in a high mIoU of 44.12, while the other methods fail at providing accurate masks. In Fig. 6, we report two examples of segmentation masks of Fun3DU on the complete point cloud of the scene, to show the complexity of the context surrounding the functional objects.

4. Implementation details

We implement Fun3DU in Pytorch and carry out all experiments on an NVIDIA A100 GPU. We use the public HuggingFace models for Owlv2[5], SAM[6] and Molmo[7].

References

[1] Tianheng Cheng, Lin Song, Yixiao Ge, Wenyu Liu, Xinggang Wang, and Ying Shan. Yolo-world: Real-time open-vocabulary object detection. In *CVPR*, 2024. 1, 2

[2] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017. 1

[3] Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, et al. Molmo and pixmo: Open weights and open data for state-of-the-art multimodal models. *arXiv preprint arXiv:2409.17146*, 2024. 3, 7

[4] Alexandros Delitzas, Ayca Takmaz, Federico Tombari, Robert Sumner, Marc Pollefeys, and Francis Engelmann. SceneFun3D: Fine-grained functionality and affordance understanding in 3D scenes. In *CVPR*, 2024. 1, 8, 9

[5] Zhening Huang, Xiaoyang Wu, Xi Chen, Hengshuang Zhao, Lei Zhu, and Joan Lasenby. Openins3d: Snap and lookup for 3d open-vocabulary instance segmentation. *ECCV*, 2024. 1, 2, 9

[6] Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. Lerf: Language embedded radiance fields. In *ICCV*, 2023. 1, 2, 9

[7] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *ICCV*, 2023. 1, 3, 7

[8] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *ACM*, 2021. 2

[9] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021. 1, 2

[10] Jonas Schult, Francis Engelmann, Alexander Hermans, Or Litany, Siyu Tang, and Bastian Leibe. Mask3d: Mask transformer for 3d semantic instance segmentation. In *ICRA*, 2023. 1

[11] Ayca Takmaz, Elisabetta Fedele, Robert Sumner, Marc Pollefeys, Federico Tombari, and Francis Engelmann. Openmask3d: Open-vocabulary 3d instance segmentation. *NeurIPS*, 2024. 1, 9

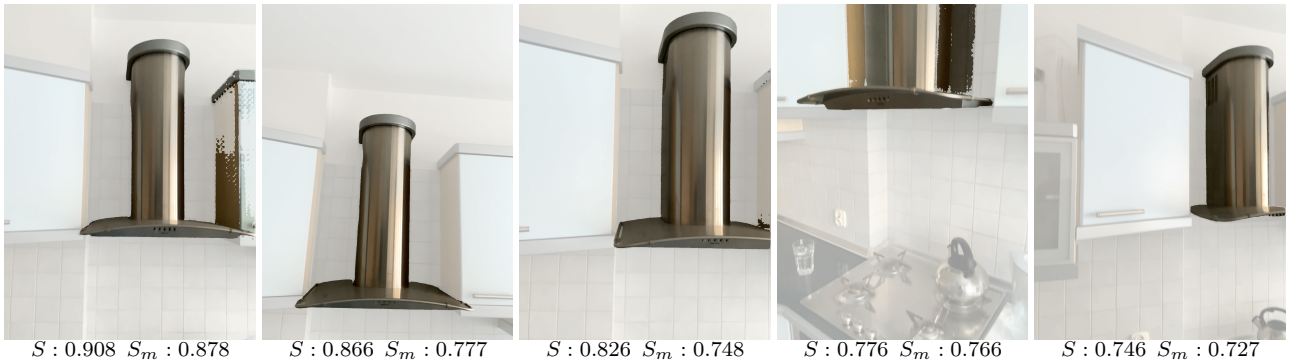
[12] Jiarui Xu, Sifei Liu, Arash Vahdat, Wonmin Byeon, Xiaolong Wang, and Shalini De Mello. Open-vocabulary panoptic segmentation with text-to-image diffusion models. In *CVPR*, 2023. 1, 2

[5] https://huggingface.co/google/owlv2-base-patch16-ensemble

[6] https://huggingface.co/jadechoghari/robustsam-vit-large

[7] https://huggingface.co/allenai/Molmo-7B-D-0924

Kitchen range hood



$S : 0.908$ $S_m : 0.878$ $S_d : 0.945$ $S_\alpha : 0.929$ | $S : 0.866$ $S_m : 0.777$ $S_d : 0.953$ $S_\alpha : 0.956$ | $S : 0.826$ $S_m : 0.748$ $S_d : 0.926$ $S_\alpha : 0.879$ | $S : 0.776$ $S_m : 0.766$ $S_d : 0.842$ $S_\alpha : 0.730$ | $S : 0.746$ $S_m : 0.727$ $S_d : 0.848$ $S_\alpha : 0.684$

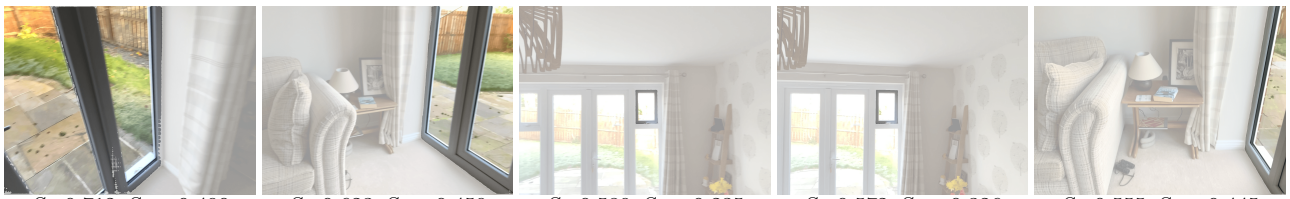
Nightstand

$S : 0.843$ $S_m : 0.715$ $S_d : 0.968$ $S_\alpha : 0.972$ | $S : 0.789$ $S_m : 0.617$ $S_d : 0.947$ $S_\alpha : 0.975$ | $S : 0.755$ $S_m : 0.798$ $S_d : 0.887$ $S_\alpha : 0.540$ | $S : 0.748$ $S_m : 0.814$ $S_d : 0.877$ $S_\alpha : 0.486$ | $S : 0.727$ $S_m : 0.843$ $S_d : 0.795$ $S_\alpha : 0.429$

TV stand

$S : 0.779$ $S_m : 0.611$ $S_d : 0.942$ $S_\alpha : 0.950$ | $S : 0.776$ $S_m : 0.634$ $S_d : 0.923$ $S_\alpha : 0.942$ | $S : 0.727$ $S_m : 0.558$ $S_d : 0.953$ $S_\alpha : 0.839$ | $S : 0.705$ $S_m : 0.621$ $S_d : 0.918$ $S_\alpha : 0.660$ | $S : 0.695$ $S_m : 0.605$ $S_d : 0.891$ $S_\alpha : 0.680$

Top right window



$S : 0.712$ $S_m : 0.490$ $S_d : 0.953$ $S_\alpha : 0.915$ | $S : 0.623$ $S_m : 0.459$ $S_d : 0.883$ $S_\alpha : 0.691$ | $S : 0.589$ $S_m : 0.235$ $S_d : 0.965$ $S_\alpha : 0.919$ | $S : 0.572$ $S_m : 0.326$ $S_d : 0.919$ $S_\alpha : 0.719$ | $S : 0.555$ $S_m : 0.445$ $S_d : 0.715$ $S_\alpha : 0.616$

Figure 2. Examples of selected contextual object masks with their scores, from highest rank (left) to lowest rank (right). We highlighted the segmented regions by alpha blending with a white background, to enhance the result visibility. On top of each view set we report the contextual object.

Open the bottom drawer of the nightstand with the red table lamp on top

Adjust the room's temperature using the radiator dial

Turn on the TV using the remote on the blue couch

Adjust the power of the humidifier

Figure 3. Examples points (in green) extracted with the VLM [3], along with the masks produced by SAM [7]. We highlighted the segmented regions by alpha blending with a white background, to enhance the result visibility.

Figure 4. Qualitative examples of Fun3DU and the baselines on split0 of SceneFun3D [4]. Point clouds are cropped around the functional object for better visualization. We report mask-level Precision (Prc), Recall (Rec), and IoU.

Figure 5. Qualitative examples of Fun3DU and the baselines on split1 of SceneFun3D [4]. Point clouds are cropped around the functional object for better visualization. We report mask-level Precision (Prc), Recall (Rec), and IoU.

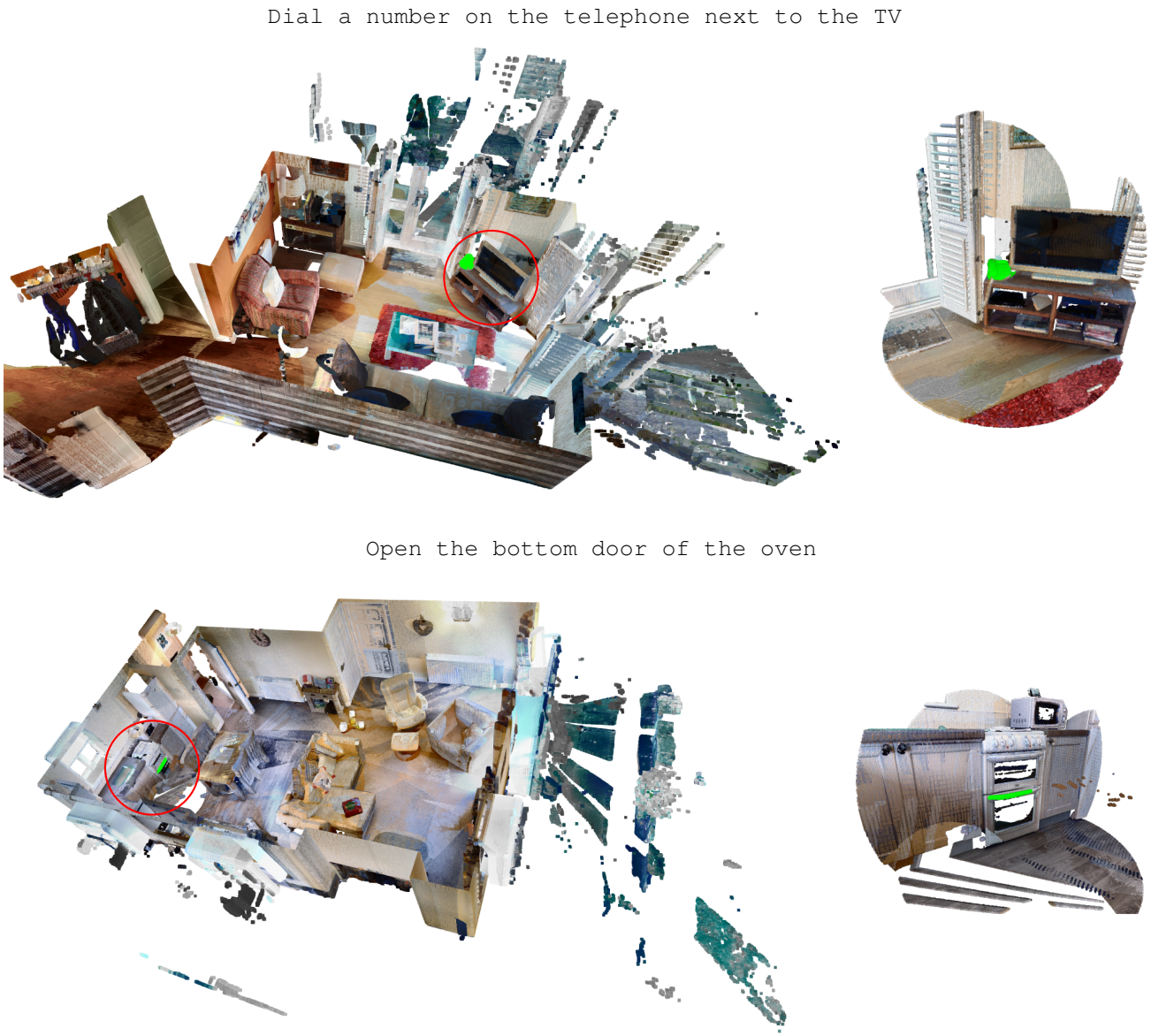


Figure 6. Qualitative results of Fun3DU on the whole point cloud (left), with detail on the portion within the red circle, that shows the segmented object (right). For better visualization, we removed the room ceilings from the point clouds.